**Brain Corporation**

# **Code Sample - Matrix Multiplication and Transposition**

## Objective

Writing a high-performance, portable linear algebra library for just transpose and multiplication of MxN matrices, without the use of standard libraries in C++.

## Assumption

### Matrix Type

For this implementation we assume a square matrix, although the same code can be easily extended to any matrix.

### Matrix Input

We use a Binomially distributed Random function to fill in the elements of the matrix. This case can be easily extended to user defined input or input from another function.

## Compiler Command

The program was built on a Ubuntu 18.04 , using Eclipse IDE 2019. This output file may be re-complied using the following command:

g++ Main.cpp -std=c++11 -pthread

## User-Defined Inputs

The following are the user defined inputs, which maybe initialized at the beginning of the program

1. **Matrix Size:** This defines the size of the matrix. For this assessment, we are assuming a square matrix. (Default Value is 4).
2. **Threads**: This defines the maximum number of threads available for the system during execution and maybe defined by the MAX\_THREADS variable. (Default Value is 4).
3. **Range of Numbers**: The variables LOW\_NUM and HIGH\_NUM maybe used to define the lowest and highest range of random numbers that matrix may be initialized. (Default value of Lowest Number is 0 and Highest Number is 100).
4. **Execution Time**: Pre-processor flags TIME\_EXECUTION maybe set to determine the execution time of each function.
5. **Display Output:** Pre-processor flag PRINT\_OUTPUT maybe set to display output of Matrix multiplication and Transposition operation.

## Matrix Structure

For this assessment, we define our own matrix structure. We use a two-dimension array of pointers, since the size of the matrix is unknown at the time of compilation.

## Solution

The program offers the following solutions:

1. **Single Thread Solution**
   1. This solution follows the conventional multiplication of 2 matrices A and B, using a single thread.
2. **Multi Thread Solution**
   1. This solution breaks down the matrix multiplication process into threads as follows:
      1. The total number of operations are calculated (MATRIX\_SIZE \* MATRIX\_SIZE)
      2. The number of operations per thread is calculated (Total Operations / Number of threads)
      3. Each thread is assigned is assigned the N number of operations.
      4. Any excess operations are assigned to the last thread.
   2. Example1

Matrix Size = 3 and Threads = 3

Total Operations = 9 and Operations/Thread =3

* 1. Example2

Matrix Size = 4 and Threads = 3

Total Operations = 16.

Thread0 and Thread1 = 5 Operations, while Thread2 = 6 Operation

1. **Transpose Solution**
   1. This solution preforms Matrix Transpose using Multi-Threading approach as defined in above.

## Note:

* If the cost of context switching is high, then performance of the multi-threading solution may not be optimal
* The pre-complied binary may be found in the debug directory